Popular Computing 30

SEPTEMBER 1975 VOLUME 3 NUMBER 9

1	2	3	4	5	6	7	....
2	1.666667	2.222222	3.074074	4.024691	5.008230	6.002743	
3	2.222222	2.037037	2.444444	3.181069	4.071330	5.027434	
4		2.444444	2.308642	2.644718	3.299039	4.132601	
5				2.532693	2.825483	3.419041	
6					2.727886	2.990803	
7						2.903164	
....							....

Main Diagonal Averages

In the cover figure, the top row of numbers and the left hand column are consecutive integers. Each subsequent cell of the array is the average of the three numbers in the cell above it, in the cell to its left, and in the cell to the northwest of it. Thus, the entry 1.666667 is found by adding 1 + 2 + 2 and dividing by 3.

The sequence of numbers along the main diagonal is to be developed. This sequence (1, 1.666667, 2.037037, 2.308642,...) is a function formed by a simple straightforward arithmetic process. If this function is to be approximated by a polynomial, what degree would that polynomial be?

The degree of a polynomial may be determined by forming a difference pattern of the functional values. For example, a difference pattern on the values of f in the array shown here indicates that, since the third differences are sensibly constant, f is of third degree:

functional values	1st, 2nd, and 3rd differences		
-12			
-24	-12		
-38	-14	-2	
-48	-10	4	6
-48	0	10	6
-32	16	16	6
5	37	21	5
72	67	30	9
172	100	33	3
312	140	40	7
498	186	46	6
736	238	52	6
1032	296	58	6

(The pattern also shows that if the seventh functional value, 5, is replaced by 6, the third differences would be absolutely constant.)

PROBLEM 104

So the Problem is this: extend the array to determine more values along the main diagonal. Difference these functional values as needed and determine the degree of the function.

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $18 per year, or $15 if remittance accompanies the order. For Canada and Mexico, add $4 per year to the above rates. For all other countries, add $6 per year to the above rates. Back issues $2 each.

Goals & Purposes

As POPULAR COMPUTING completes two and a half years of publication, it seems appropriate to state the magazine's goals:

1. To encourage computing for fun. This is the chief goal, in the belief that computing is fun, and that solving a difficult problem by computer (when the computer is the proper tool to use) is richly rewarding. The solution to a problem is often not important; it's getting to that solution that constitutes the fun.

   We can also have fun along the way by not taking ourselves too seriously. Computing is certainly unusual among technological disciplines in having produced a steady stream of humor, much of it poking fun at our own traits.

2. To demonstrate that what to compute is as important as how to compute; that not every problem situation involving the processing of information is a computer problem.

3. To emphasize that validating the logic and coding of a computer problem solution is also vital; that no program should be put into production without thorough testing.

4. To foster good program design; that is, to encourage clear thinking about a proposed solution before any coding is attempted. The specific tools for this purpose are a matter of taste more than anything else (although current thinking on the matter differs on this point, and the opposing views will be presented too). If flowcharts furnish for you a way to get the logic straight, then you should use flowcharts. If you prefer the pseudocode approach, more power to you. There are those who prefer to organize a solution in terms of decision tables; still others like narrative flowcharts. As we see it, all these tools do much the same thing--they clear up the logical troubles before those troubles that are indigenous to coding can be introduced, and they aid in tracking down and correcting those troubles after they occur.

Our slogan is "The way to learn computing is to compute." The implication is that continued devotion to the art will produce improvement in it, and the largest area for improvement by all of us is that of program design.

5. To show that generations of workers have devised brilliant ways of coercing better answers from data with less work--and these ideas are all still good and useful. It all comes under the category of

> Cat, there are more ways than one to skin a.

There is always a better way--and a still better way after that.

6. To offer as many new problems for computer solution as can be devised. This goal is aimed at teachers of computing, who are always short of good problems for class use. On any campus, the half life of a good new problem is about four semesters, after which the problem loses its charm and identical solutions begin to appear. The life of a problem can be extended by altering its parameters, or by forcing a change in programming language, but a source of new problems is still of great value.

> The more problems that are available (and that are pre-tested as suitable for student use), the more likely it is that one can be found that will excite any given student.
>
> We have averaged well over three new problems per issue. Every time that an issue has appeared with more than one new problem in it, the reaction from readers was the same; namely, "Why did you publish such a trivial (useless, uninteresting, poor, dull, etc.) problem as A, when problem B is so good (interesting, unusual, novel, challenging, appealing, etc.)?" Invariably, two successive people would have opposing views as to which was problem A and which was problem B. That's great: the best problem to work on is the one that _you_ find challenging.

7. To focus some attention on the built-in pitfalls and booby traps of our languages and operating systems; to call attention, as vividly as possible, to the fact that all computer languages are only a stepping stone to the language that will be executed, and the latter is usually the binary language of the machine itself. The high level languages do provide great power and do make programming easier, and do tend toward machine independence, but at the price of interdiction; that is, the interposing of thousands of instructions (and hence decisions) between the user and the machine. Those thousands of instructions do things _to_ you as well as _for_ you, and not all of them are known. None of the above is to be construed as an argument in favor of assembly language coding.

8. To encourage more high precision arithmetic (the list in PC21 of all the arithmetic that has been done to 500 digit precision or greater is rather small), and to point out situations in which high precision arithmetic is vital in order to obtain _any_ precision in the desired result.

For example, the innocent looking Gear Problem (in PC28) demands arithmetic of at least 12 significant digits in order to have the solution come out in the correct year, much less to pinpoint the day and hour.

9. To present known results and tables (for example, the continuing N-series). An old problem becomes new when we can extend the limits of knowledge about it. Breaking existing records is part of the fun of computing.

10. To demonstrate the "unpredictable computer" concept; namely, that the result of running any non-trivial program is difficult to predict and frequently holds surprises for the person who wrote it. Indeed, it is not difficult to show that this principle can apply even to trivial programs, and students of computing should be aware of it.

11. To help dispel the belief that computers are only overgrown fast adding machines or bookkeeping machines. They can be all of that, but the ability to take courses of action that depend on the data (whether that data is part of the input, or is derived) is a new tool. As J. Presper Eckert put it, "The computer may be the first general purpose tool in a university since the library."

12. To promote the art of computing, to the end that some day it may take on some of the aspects of a science and its practitioners may consider themselves members of a profession. □

The 196 Problem

The palindrome problem has been discussed at length in the literature. Start with any positive integer. Reverse its digits and add the two numbers. Repeat the process with the sum until a number is reached that reads the same forwards and backwards. For most integers, this will occur in a finite number of steps, and it has been conjectured that this is true of all integers. For all but 249 integers less than 10,000, the number of stages needed to form a palindrome is 24 or less. The calculations for the integer 89 are shown on this page.

Martin Gardner has traced the problem back to the 1930's. Very early in the game, it was noticed that, of the integers that do not become palindromes within 24 stages, most of them do not get there within 100 stages, and apparently will not get there at all. The smallest integer of concern is 196, and a great deal of computation has been done on that number. For example, a run to 8225 stages by Thomas Sardi produced a 3433-digit number which begins and ends:

1775796080651824 ... 2418255190588676.

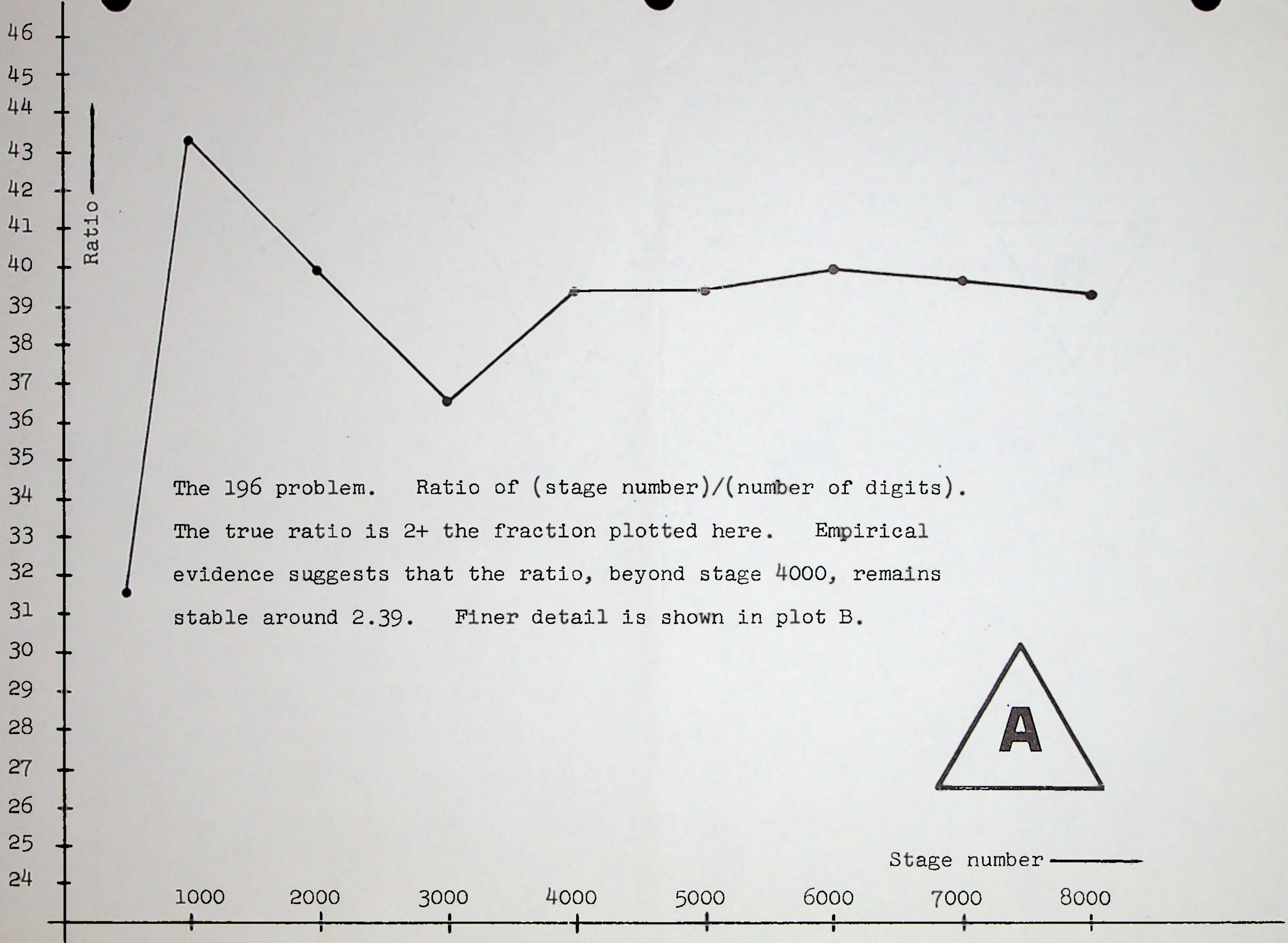
The proof in Simmons' paper that the probability of a randomly selected integer being palindromic approaches zero as the number of digits in the integer increases becomes pertinent. Although a proof is still needed, it would seem that the conjecture cited above is not true.

While the effort to extend the 196 calculation to fantastic heights is interesting in itself, a new problem has emerged that is much more interesting. For most sequences of numbers that grow, the growth rate (in decimal digits) is constant. For example, a table of powers of 2 gains one digit for each

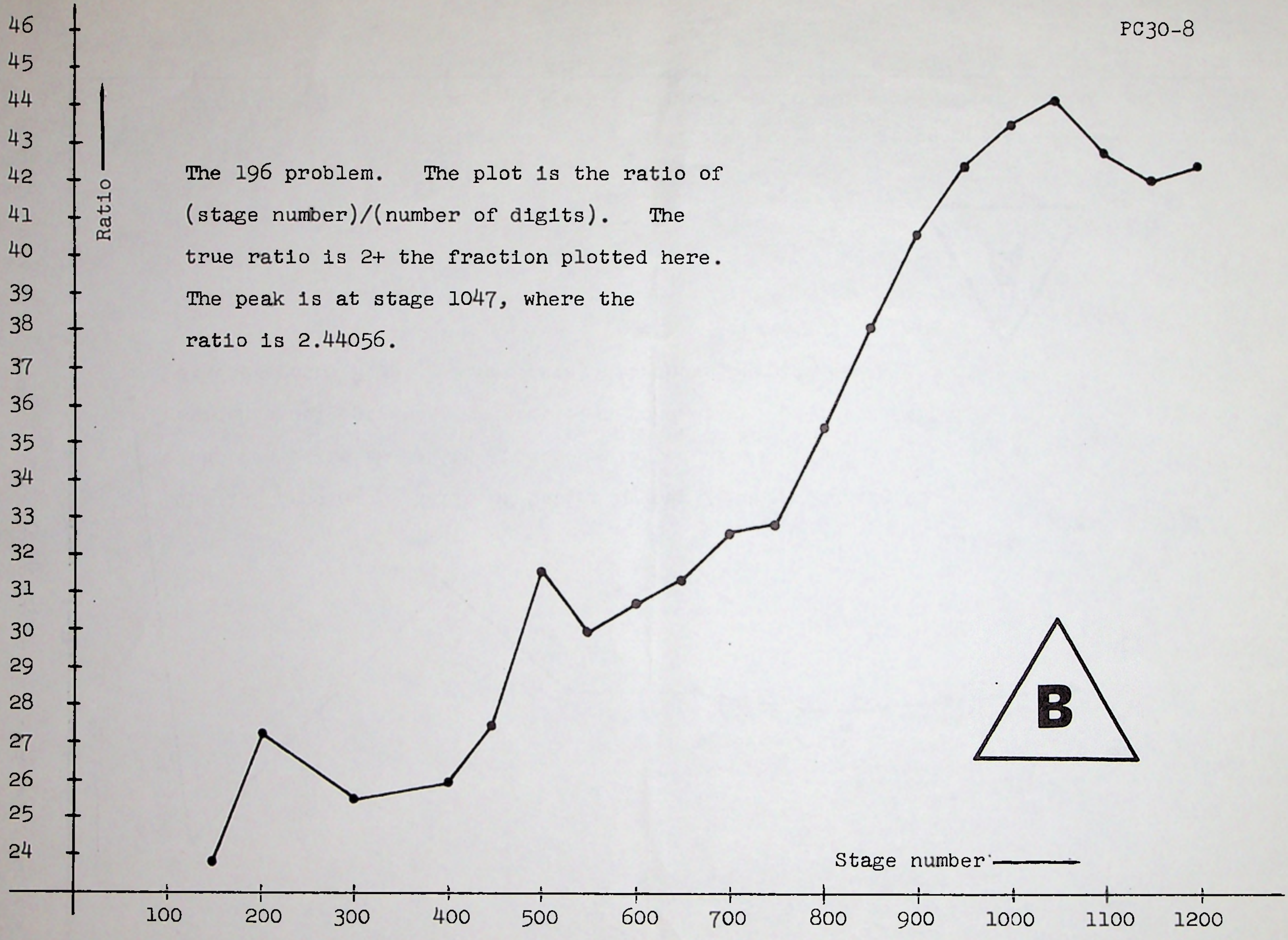
$$1/(\log_{10}2) = 3.3219 \text{ stages.}$$

Similarly, the Fibonacci sequence gains a digit about every 5 stages (the 500th term has 105 digits), and this growth rate holds steady. The successive stages of the 196 problem, however, exhibit a curious growth rate. See Figure A.

89	
98	
187	1
781	
968	2
869	
1837	3
7381	
9218	4
8129	
17347	5
74371	
91718	6
81719	
173437	7
734371	
907808	8
808709	
1716517	9
7156171	
8872688	10
8862788	
17735476	11
67453771	
85189247	12
74298158	
159487405	13
504784951	
664272356	14
653272466	
1317544822	15
2284457131	
3602001953	16
3591002063	
7193004016	17
6104003917	
13297007933	18
33970079231	
47267087164	19
46178076274	
93445163438	20
83436154439	
176881317877	21
778713188671	
955594506548	22
845605495559	
1801200002107	23
7012000021081	
8813200023188	24



The 196 problem. Ratio of (stage number)/(number of digits). The true ratio is 2+ the fraction plotted here. Empirical evidence suggests that the ratio, beyond stage 4000, remains stable around 2.39. Finer detail is shown in plot B.



The 196 problem. The plot is the ratio of (stage number)/(number of digits). The true ratio is 2+ the fraction plotted here. The peak is at stage 1047, where the ratio is 2.44056.

The growth rate (expressed as the number of stages per digit), after some erratic behavior near the origin, rises to a peak at stage 1047, and seems to stabilize around 2.39 beyond stage 4000. A blowup of the critical area of Figure A is shown in Figure B. Thus, when someone claims to have carried the 196 calculation to 10,002 stages, resulting in a 3798-digit number, the declared ratio is 2.633, which seems unlikely. The behavior of this ratio is mysterious, and calls for further investigation. The integers 196, 295, 394, 493, 592, 689, 691, 788, 790, 887, 986 all lead to the number 1675 after one or two stages, so that at the moment there is no other known candidate for a non-palindromic integer. When one is found, it will be interesting to see whether or not it also exhibits a peak in the ratio.

References

Ahl, David, "Follow-up on Palindromes," Creative Computing, May/June 1975, pg. 18.

Francis, Darryl, Games & Puzzles, June 1974, pg. 29.

Gardner, Martin, "Mathematical Games," Scientific American, August 1970, pgs. 110-114.

Koetke, Walter, "Palindromes; For Those Who Like to End at the Beginning," Creative Computing, Jan/Feb 1975, pgs. 10-12.

Simmons, Gustavus, "Palindromic Powers," Journal of Recreational Mathematics, April 1970.

Trigg, Charles W., "Palindromes by Addition," Mathematics Magazine, Jan 1967.

Starting value	100000	1000000	9000000	to reach this limit
1	4741	39014	303507	
3	4606	38270	294912	
7	4737	39014	303503	
20	4734	39011	not calculated	

Growth rate for the sequence SLOWGROW

For an example of regular series growth, consider the sequence of numbers (SLOWGROW) in which each term is the preceding term plus the sum of its decimal digits. Starting with, say, 7, the sequence is given on this page.

The table on the preceding page shows the number of terms for four different starting values, to reach each of three limits. Each of these sequences is independent of each other; the point is that most growth sequences have a regular growth rate.

7
14
19
29
41
46
56
67
80
88
104
109
119
130
134
142
149
163
173
184
197
214
221
226
236
247
260
268
284
298
317
328
341
349
365
379
398
418
431
439
455
469
488
508
521
529
545
559
578
598
620
628
644
658
677
697

N-SERIES 30

Log 30	1.477121254719662437295027903255115309200128864190696
Ln 30	3.401197381662155375413236691606889912248592046451522
$\sqrt{30}$	5.477225575051661134569697828008021339527446949979833
$\sqrt[3]{30}$	3.107232505953858866877662427522386362854906829067422
$\sqrt[5]{30}$	1.974350485834819842672836172408453182682267248095355
$\sqrt[10]{30}$	1.405115826483646095677425893053899523002230031699556
$\sqrt[100]{30}$	1.034596994728644014201313027554269880441883669325276
e^{30}	10686474581524.46214699046865074140165002449500547305 49902229114921084529447871315325380577
π^{30}	821289330402749.5815865035854340488221964711968781287 2744033711027572453906473704466409354
$\tan^{-1}$ 30	1.537475330916649422075173902618357495499458182493186

Printed by Argold Press Inc.
16921 Ventura Boulevard, Encino

The accompanying table extends the one given in PC25-6. Selected pairs of terms in the Fibonacci sequence are shown. The term number is counted as follows:

Term	1	1	2	3	5	8	13	21	34	55	89	144	...
Term number	1	2	3	4	5	6	7	8	9	10	11	12	...

The calculations were done on the Altair 8800, using the overall logic of flowchart P and the detailed logic of flowchart Q. One decimal digit was carried in each word of the machine.

> Herman P. Robinson reports "On page 6 of PC25 the last four digits of the ratio are in error. I found the ratio to be the same as the golden mean to at least 45D. In fact, if the two Fibonacci terms (500th and 501st) are correct, the ratio will give the golden mean to about 210 significant figures, which follows from the theory of continued fractions."

599	6825139109610030996497844604508742030702560685972243832 3487946038808981838031799984351367205238184363410615527949660089420401
600	110433070572952242346432246767718285942590237357555606380008891875277701705731473925618404421867819924194229142447517901959200
699	540599366663078885853712245240404795641933408471282749908273500633697524067672844867129081639663420912107124987546834669159043581536363174426394 26
700	874708149557528462039784130175713273423672409676973810742304325925275019112903776556282271508784273316531933691091936723307775279437181691051242 75
999	2686381002448535938614672720214292396761660931898695234012317599761798170024788168933836965448335656419182785616144335631297667364221035032463485041037768036733415117289916972319708276398561576445007847417462 6
1000	4346655768693745643568852767504062580256466051737178040248172908953655541794905189040387984007925516929592259308032263477520968962323987332247116164299644090653318793829896964992851600370447613779516684922887 5

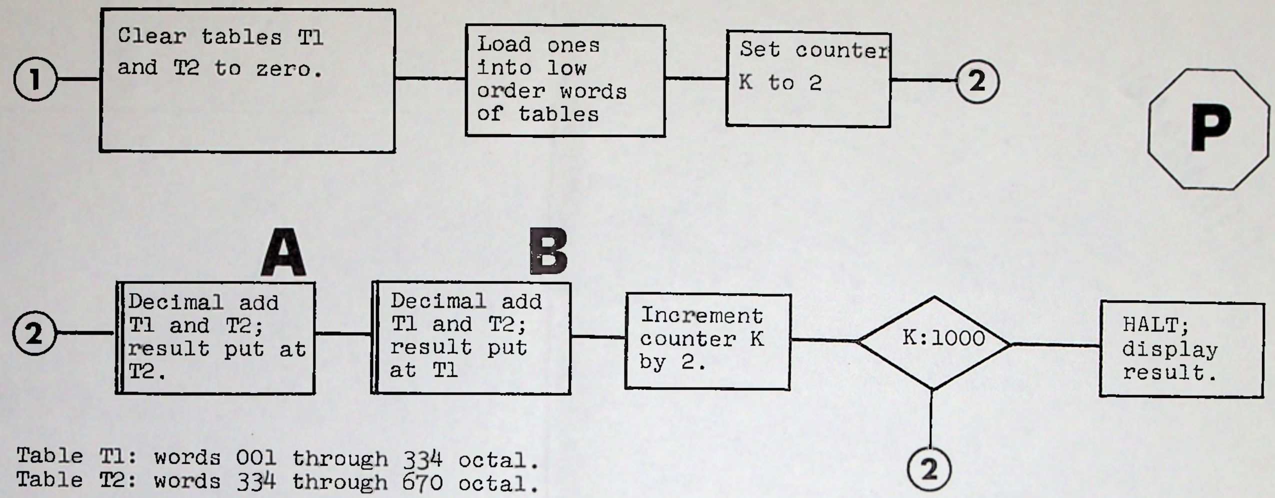
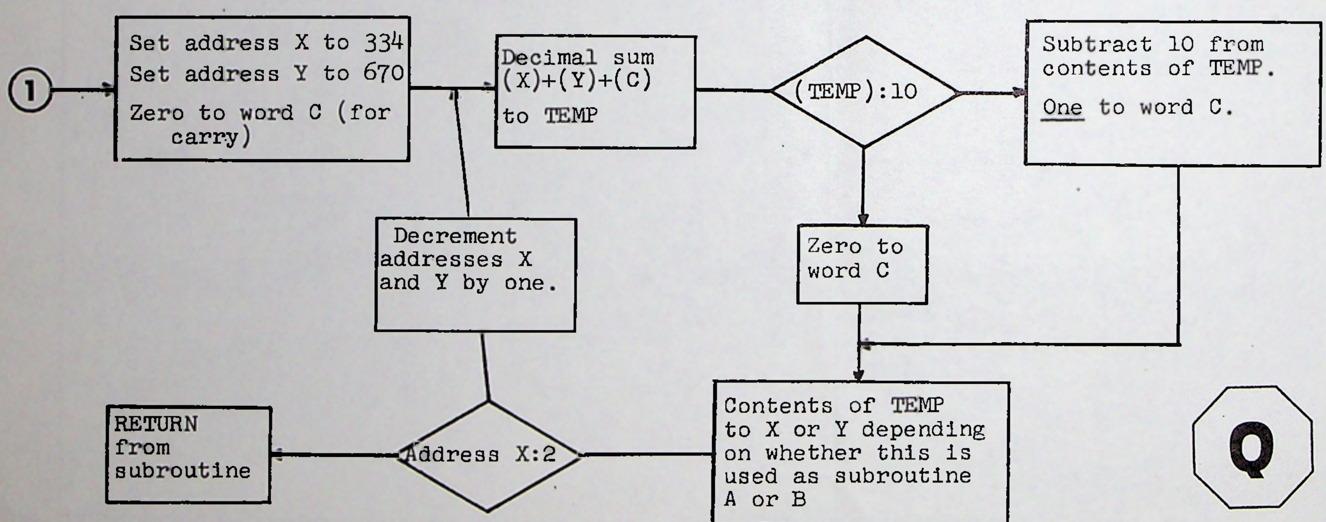


Table T1: words 001 through 334 octal.
Table T2: words 334 through 670 octal.



Subroutine for decimal addition of tables T1 and T2.

In the "Speaking of Languages..." column in October 1973 (Issue No. 7), Robert Teague presented the Change Maker problem. The input is to be an amount tendered for a purchase, and the amount of the purchase; the output is to be the correct change, in the smallest number of U.S. bills and coins. A solution was given in February 1974 (Issue No. 11) written in ANSI Fortran in nine statements.

M. E. Sandfelder, IBM Poughkeepsie, offers the following solution in APL for the same problem:

```
∇CHANGE[⎕]∇
    ∇ R←CHANGE A;KK;TITLE
[1]   TITLE←6 13⍴'DOLLAR BILLS=HALF DOLLAR =QUARTER      =DIMES        =NICKEL       =PENN
IES      ='
[2]   'AMOUNT TENDERED= ',(2⍕A[1]),'      AMOUNT OF PURCHASE= ',(2⍕A[2]),' AMOUNT OF CHA
NGE= ',⍕0⌈-/A
[3]   R←(0≠KK)⌿TITLE,⍕6 1⍴KK←(3↑0 2 2 25⊤A),0 2 5⊤25|A←0⌈⌊100×-/A
    ∇
```

throws 0 coins
joins to title
converts to character data
reshapes into array
takes only dollars, H, & Q
encodes to dollars, H, Q, & pennies
encodes to D, N, P
25 residue throws out quarters
tests to make sure zero
forces integer
multiplies by 100
finds difference in A

Mr. Sandfelder comments: "The printout shows a two line APL solution which not only is shorter than the Fortran solution, both in lines, two for APL and nine for Fortran, but more significantly the APL solution requires 211 characters versus over 760 characters for the Fortran solution. In addition, the APL solution has two extra features. One is that the solution prints only the coins needed and if you purchase more than you tender you do not receive any change.

"One other comment. There is a tendency among some APL enthusiasts to write 'APL one-liners' at all costs. The given solution is not a forced one-liner, but a natural result of the power and syntax capabilities of the APL language."

In this connection, Jean Sammet, writing on "Computer Programming" in the <u>1973 Yearbook of the Encyclopedia of Science and Technology</u>, said "One development that verges almost on the border of being called a phenomenon is the increased interest and use of APL/360 and its versions on other computers. The adherents of this language support it with a vigor that is unlike anything ever seen in the history of programming languages."

Another problem is the following, from the SUNY-Binghamton Computer Center Newsletter for February 1975: Find the indices of the smallest element in a matrix. The most general solution, from John Gaboury, Sun Life of Canada, is reproduced here, together with an analysis furnished by G. Truman Hunter, IBM Poughkeepsie:

```
⍉ 1+ Z⊤ ¯1+ ( ,M= ⌊/ ,M )/ ⍳ ×/ Z←⍴M
```

gets coordinates of matrix

gets total number of elements of matrix

builds integer string 1, 2, ..., to help identify each element of matrix

converts matrix to single string of numbers (vector)

finds all the smallest values in string

puts 1 for smallest values, 0 for all others

selects one dimensional index of all smallest values in string

changes index base to zero, preparing to convert back to matrix form

converts back to row and column index form for location of smallest element of matrix

converts back to base 1 for row and column indentification

converts to regular row and column notation, since previous answer gave column and row oriented result

Mr. Hunter writes: "I have written an analysis of the code that shows how the different actions are all cascaded along the line, so you don't have to stop at intermediate results. This also shows why APL code can be so compact." □

Calculator Ratings

In Issue 10 there was given a rating scale for desk and pocket calculators:

1.	Floating point arithmetic	200
2.	Scientific notation	200
3.	Number of digits, D, in calculations	100(D-8)
4.	Number of digits, D, in display	20(D-8)
5.	AC operation only, no points	
	DC operation (i.e., portable, batteries)	50
	AC and rechargable batteries	50
6.	Constant multiplier and divisor	50
7.	Addressable storage, per word	100
8.	Square root function	100
9.	Reciprocals	50
10.	Trigonometric functions and inverses	300
11.	Logarithm and antilogarithm functions	300
12.	Adjustable rounding	50
13.	Additional functions, per function (Other than trivial. A feet-to-meters function is trivial, for example.)	200
14.	Variable fixed point	100
15.	Additional features, per feature (Other than frills)	200
16.	False claims, per claim	-100

Total points divided by retail price = index value

At that time, the ratings of machines then available went up to around 7. Since then, because of the increased power of the pocket machines and the dramatic drop in prices, the ratings have gone up sharply. In the current market, there are machines offered that have a rating of over 30 on our scale, due to sale prices.

The rating scale did not include provision for programming capability (such as on the Compucorp machines, the HP-55, and HP-65). Simple programming capability should add 500 points; capability for storing programs should add another 1000 points. Other exceptional features can be rated, such as:

printing capability	1000
compound interest functions	300
standard deviation function	200
recognized and stable brand	300
meaningful warranty	200

The rating scale should not be considered as an attempt to reduce the continuum of calculators to a one-dimensional vector. It can provide an objective means for comparing machines of the same class. The user of the scale is urged to set his own point values before applying the scale to any machine.

The various trends are becoming clear. The simple 4-function machines are now extremely cheap, selling widely for under $20. By and large, 6-digit machines are disappearing, so that 8-digit floating point is the standard. Middle class machines add some functions and one word of storage, for a price range centering around $50. Upper middle class machines (around $100) add trig and log functions and scientific notation.

And then there are the high class machines. The latest of these is the Texas Instruments' SR-51 which has some interesting new features:

> A random number generator. Numbers in the range 00-99 are generated, with the starting value supplied by the machine.
>
> Factorials, permutations, and combinations.
>
> Hyperbolic sine, cosine, and tangent, and their inverses.

On our scale, the SR-51 has a rating of 29.

The SR-22, also from TI, is a special purpose desk machine that is unique. The machine works in hexadecimal notation, and intermediate and final results can be displayed in decimal or octal. Thus, to take the square root of $1000)_{10} = 3E8)_{16}$, the Newton algorithm proceeds:

$$
\begin{aligned}
3E8/20 &= 1F.4 \\
(20 + 1F.4)/2 &= 1F.A \\
3E8/1F.A &= 1F.9ED \\
(1F.A + 1F.9ED)/2 &= 1F.9F6 \\
3E8/1F.9F6 &= 1F.9F7C9 \\
(1F.9F6 + 1F.9F7C9)/2 &= 1F.9F6E4994 \\
3E8/1F.9F6E4994 &= 1F.9F6E4991
\end{aligned}
$$

and $1F.9F6E4991)_{16} = 31.6227766)_{10}$

If the user works in decimal or octal, there are long waits after every key depression (other than digit entry) while the machine converts to hex.

The SR-22 sells for $250. □

Book Review

Fortran to PL/1 Dictionary PL/1 to Fortran Dictionary
by Gary DeWard Brown, John Wiley & Sons, 204 pages, 1975, $10.95.

Reviewed by Nadine Malcolm

This dictionary speaks to a rather limited audience: Fortran programmers who wish to learn PL/1 and PL/1 programmers who wish to learn Fortran. The presentation is straightforward: each chapter covers a single topic; e.g., basic language statements or input/output. Each page is divided into two columns; the left column explains a feature of Fortran, and the right column explains the corresponding feature of PL/1. PL/1 features that have no analog in Fortran are explained in an appendix. Language features are easy to find, and the explanations are clearly and concisely written. For quick reference there is even a language summary showing the corresponding language features without explanations. There is a set of problems at the end of each chapter. Since they tend to cover the most easily misunderstood language features, and since solutions are given for most of them, they greatly enhance the book's self-instruction potential.

Fortran to PL/1 is not without flaws. The first problem is that when Mr. Brown speaks of Fortran, he means not ANSI standard Fortran, but IBM 360/370 Fortran. This seriously limits the book's usefulness; PL/1 programmers may want to learn Fortran simply because they are switching from IBM machines to another manufacturer's equipment. The book also has a few technical errors. For example, it states "Fortran subroutines must have at least one argument," although the Fortran standard explicitly states that arguments are optional. Fortunately, such errors are relatively minor.

This dictionary could be quite useful to experienced programmers, especially those using IBM equipment, who need to learn a new language. It is not intended to be a beginning text for teaching programming using Fortran or PL/1, nor does it seem appropriate for this purpose. □